NORD SECURITY

Network Interfaces

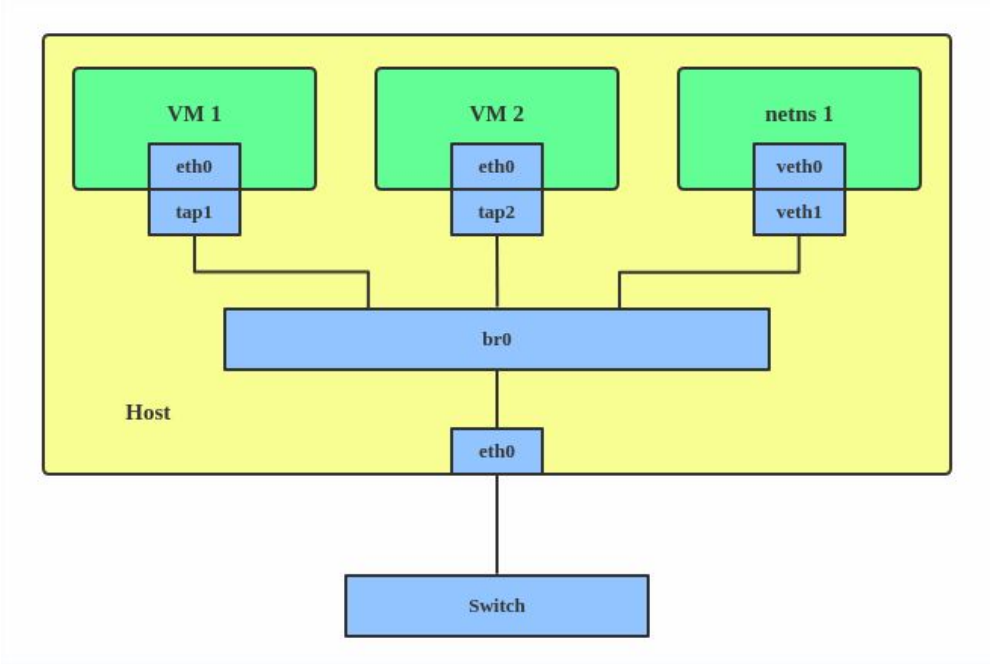
Types of interfaces

- Physical
  - Eth, wlan -> old
  - En0, anp1 -> new
- Virtual
  - Bridge
  - Bond
  - Vlan
  - tun/tap

Virtual interfaces

Bridge

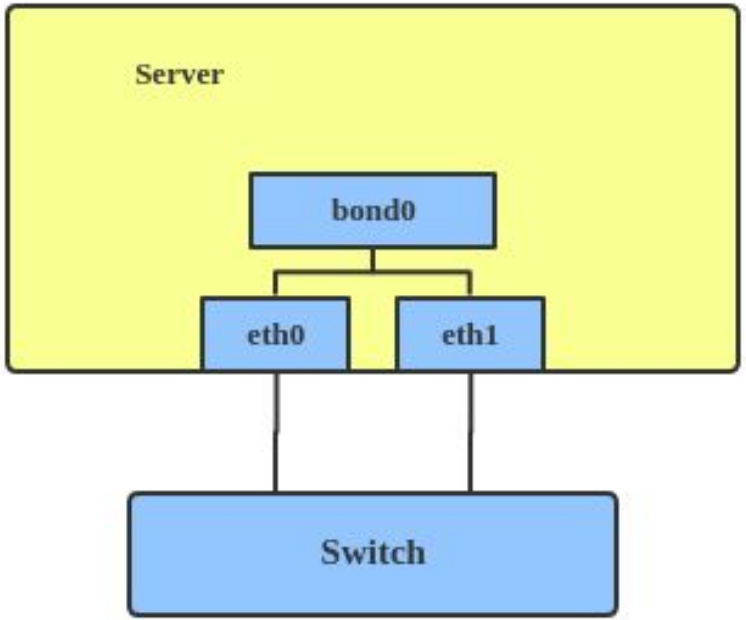
- Acts like a network switch
- Forwards packets between connected interfaces.
- Example usage: establish communication Between VM's, containers and host



Virtual interfaces

Bond

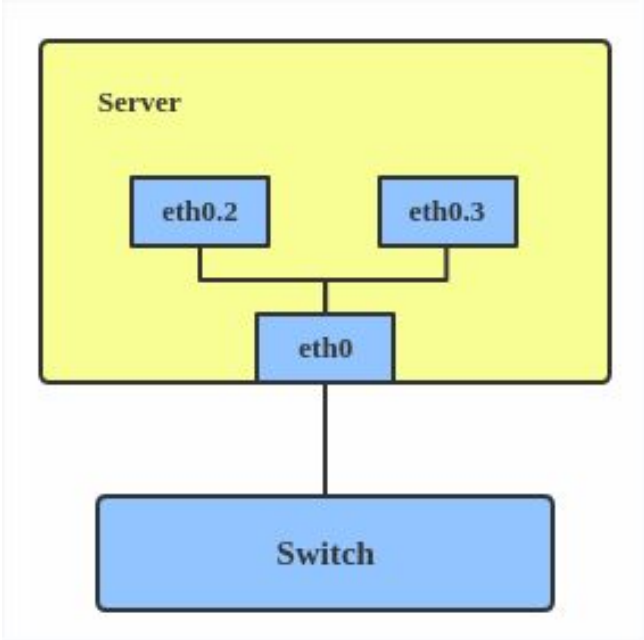
- Aggregates multiple interfaces into 1 logical interface.
- Usage: For hot standby or load balancing



Virtual interfaces

Vlan

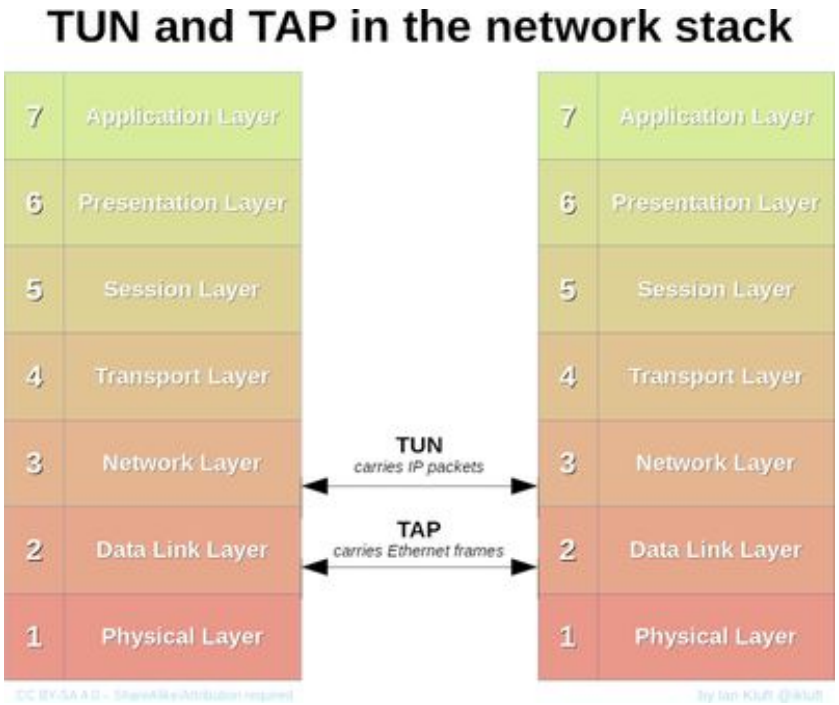
- Creates a virtual lan device
- Separates broadcast domains by adding Tags to network packets
- Used when you want to separate in VM’s or hosts.



Virtual interfaces

TUN/TAP

- Used in VPNs
- Tunnels packets between kernel and userspace
- TUN and TAP cannot be used together



Tools to create interfaces

Iproute2

- Replaces a number of legacy tools like ifconfig, route, netstat
- Includes a suite of tools to manipulate interfaces

Legacy	Replacement	Usage
ifconfig	Ip addr, ip link	Address and link config
route	Ip route	Routing tables
netstat	Ip route	Various networking stats
arp	Ip neigh	Neighbours
brctl	bridge	Handle bridges

Some common commands

Show all addresses

ip addr show

Show address for a single interface

ip addr show <interface>

Delete an address

ip addr delete <address>/<prefix> dev <interface>

Some common commands

View all routes

ip route show

View route to a network and all its subnets

ip route show to route <address>/<mask>

Add route via interface

ip route add <address>/<mask> dev <interface>

Change a route

ip route change <address>/<mask> via <address>

Delete a route

ip route delete <address>/<mask> via <address>

Some common commands

Show all link information

ip link show

Show specific link information

ip link show dev <interface>

Bring a link up or down

ip link set dev <interface> <up/down>

Create a bridge

ip link add name <bridge-name> type bridge

Add interface to a bridge

ip link set dev <interface> master <bridge>

Most common commands

Create a bond

ip link add name <bond> type bond

Add interface to a bond

ip link set <interface> master <bond>

Create a tun interface

ip tuntap add dev <interface> mode <mode>

Cheatsheet: https://paulgorman.org/technical/linux-iproute2-cheatsheet.html

Exercise

Ex # 1
Run ls /sys/class/net on client1. What do you see ?

Ex # 2
Create a bonded interface on client1

```
ip link add bond0 type bond mode balance-rr
ip link set eth0 down
ip link set eth1 down
ip link set eth0 master bond0
ip link set eth1 master bond0
ip link set eth0 up
ip link set eth1 up
ip link set bond0 up
ip addr add <ip-addr> dev bond0
ip addr
```

Ping server1 (either IF) using bond0 as interface. What is happening ?

Sockets

- Socket is a special file for inter-process communication
- Bidirectional communication pipe that enables 2 processes to exchange information
- Typical use case -> client-server model
- 4 Types of sockets
  - Stream oriented
  - Datagram oriented
  - Raw sockets
  - Sequenced packet sockets

Types of sockets

Stream oriented sockets SOCK_STREAM

- Reliable connection based communication channel
- Mainly uses TCP/IP as its network stack
- Establishes persistent connection between 2 processes
- Use case
    - Web servers
    - Email servers

You can look at current listening TCP sockets by:

```
ss -tln
```

Types of sockets

Datagram oriented sockets SOCK_DGRAM

- Supports connectionless UDP
- Each packet is an independent datagram
- Arrival and integrity not guaranteed
- Use case
    - VoIP
    - Online gaming

You can look at the current UDP sockets by running the following command

```
ss -uln
```

Types of sockets

Raw sockets SOCK_RAW

- Enables to access the underlying network protocols directly
- Allows to exchange data at the lowest level, avoiding transport layer formatting
- Used when you need high level control over communication

Types of sockets

Sequenced sockets SOCK_SEQPACKET

- Allows packets to be managed at the network layer
- Offer middle ground between stream and datagram sockets
- Not often used

How sockets work

Each socket is described by domain (eg IPv4/IPv6) and type (eg UDP/TCP)

All the communication is done via the socket API

- Establish and manage connections
- Transfer data between processes or machines

How sockets work

Let’s say we want to establish a connection-oriented server-client socket

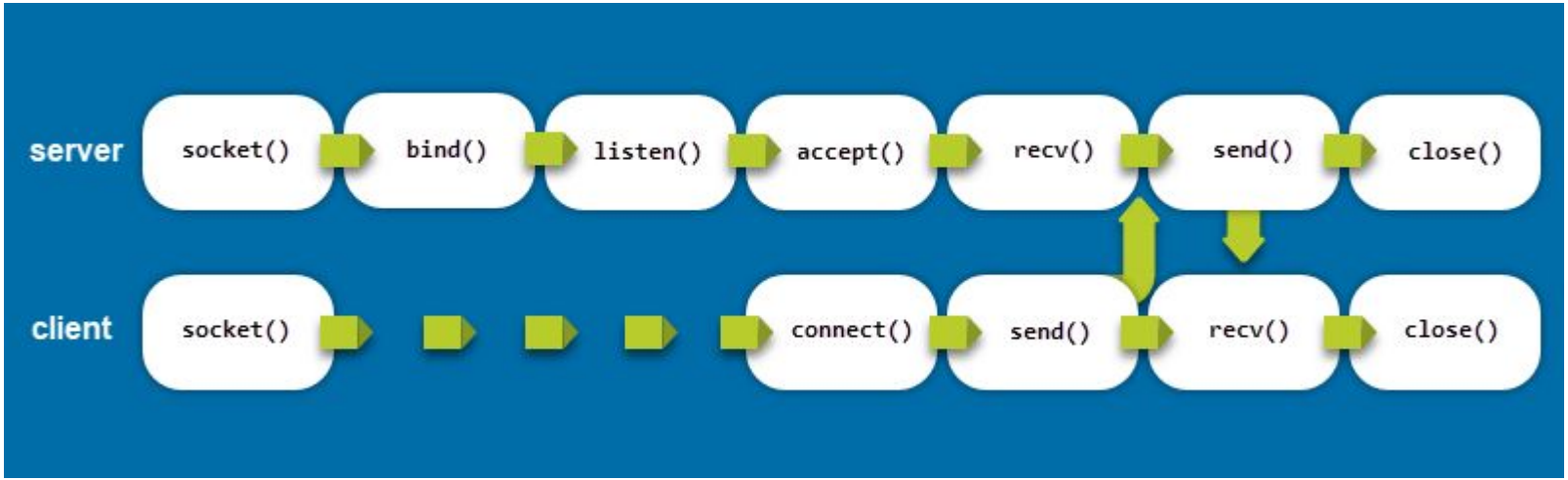
On the server side

- Bind() - binds a socket to a network address and port
- listen() - tells the server to wait for incoming connections
- accept() - receives client connections
- read() and write() - communicate with remote endpoint once connection has been established
- close() - closes connection

On the client side

- connect() - connects with the server
- send() and recv() - send and receive data
- close() - closes connection

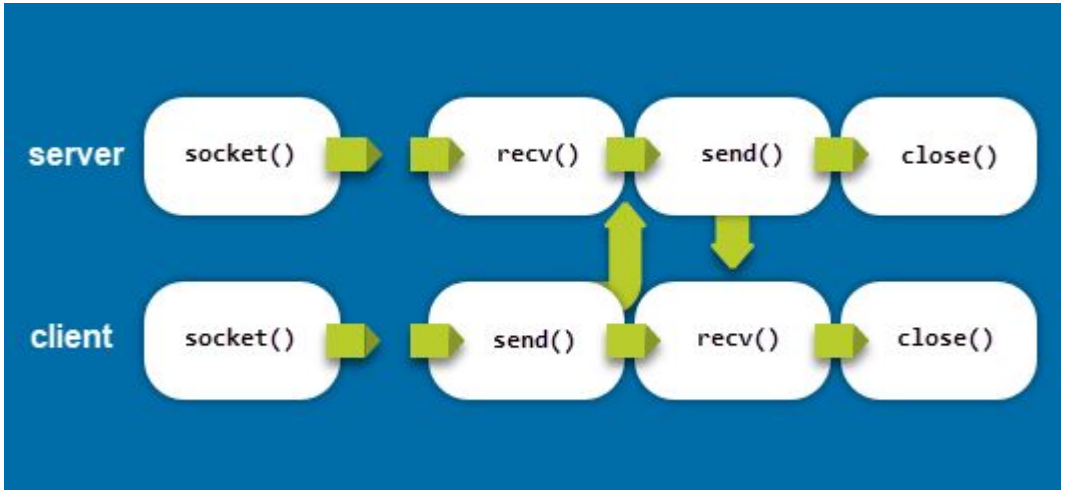
How sockets work



How sockets work

Datagram socket connection

- No need to wait for and accept connections



Exercise

- Write a small client-server application in python sending “Hello world!”
    - Client1 as client
    - Server1 as server
    - Using SOCK_STREAM
    - Port as 12345

Note : before beginning remove the bond0 interface from previous exercise

ip link del bond0
ip link set eth0 up
ip link set eth1 up

SERVER

```
import socket

def main():

    # Create a TCP socket
    with socket.socket(socket.AF_INET, socket.<socket-type>) as server_socket:
        # Bind the socket to the address and port
        server_socket.bind((<host>, <port>))

        # Listen for incoming connections
        server_socket.listen()

        # Accept incoming connections
        connection, address = server_socket.accept()

        # Receive data from the client
        data = connection.recv(1024)
        print("Received:", data.decode())

if __name__ == "__main__":
    main()
```

CLIENT

```
import socket

def main():

    # Create a TCP socket
    with socket.socket(socket.AF_INET, socket.<socket-type>) as client_socket:
        # Connect to the server
        client_socket.connect((<host>, <port>))

        # Send data to the server
        message = "Hello world!"
        client_socket.sendall(message.encode())

if __name__ == "__main__":
    main()
```

Homework

- Write a small client-server application in python sending “Hello world!”
    - Client1 as client
    - Server1 as server
    - Using SOCK_DGRAM
    - Port as 12345